Faculdade de Tecnologia Rubens Lara

RELATÓRIO - REDE NEURAL E PROBLEMA DE MINIMIZAÇÃO : Previsão de precipitação por MLP

3º Ciclo - Ciência de Dados

Igor Silva de Carvalho

Santos, 2025

Sumário

1 INTRODUÇÃO

Problemas de grande complexidade, das quais a necessidade de flexibilidade se demonstra de grande importância, são muitas vezes difíceis serem resolvidos com modelos de regressão tradicionais. É nesse contexto que algoritmos como redes neurais são de grande utilidade. Redes neurais são programas de *Machine Learning* (ML) que procuram fazer decisões de forma similar à mente humana (IBM, 2021).

Neste relatório será descrito as operações teóricas das técnicas por trás da criação de uma rede *Multi-Layer Perceptron* (MLP) para a previsão de precipitação em uma dada hora a partir de dados pontuais contemporâneos à previsão. Os dados utilizados foram solicitados por meio do Instituto Nacional de Meteorologia (INMET) para a estação meteorológica automática A701 em Mirante de Santana - SP[1].

Para a criação da rede neural foram utilizados, na etapa de treino, dados horários de variáveis meteorológicas do período compreendido entre **01/01/2010** até **25/05/2025**. A escolha dessa amostra de dados foi feita levando-se em conta a relativa proximidade, em relação a Santos, da estação e a acima de tudo a quantidade de dados disponíveis pela estação.

Para a utilização desses dados, estes foram - por meio da linguagem de programação Python - primeiro limpados através da biblioteca Pandas, para então serem explorados, e plotados (grafados) - por meio da biblioteca Matplotlib e Seaborn -, e finalmente foram utilizados para o treinamento e avaliação da rede neural por meio da biblioteca TensorFlow como *backend* para a *Application Programming Interface* (API) Keras. O modelo criado contém 3 camadas: de *Input* com 6 neurônios, oculta com 4 neurônios e de *output* com 1 neurônio. Após o treino, o modelo atingiu uma acurácia de aproximadamente 93% em suas previsões.

Este trabalho procura relatar o processo de síntese dos algoritmos, que culminou na criação de um repositório Github[2] contendo: o *dataset* utilizado; a exploração dos dados; o treino da rede neural; a avaliação e verificação da acurácia de previsão do modelo.

2 REFERENCIAL TEÓRICO

A fim de melhor entender o conceito de Rede Neural, assim como seus componentes, a seguinte seção propõe explicar os fundamentos teóricos por trás dos conceitos abordados, assim como os parâmetros utilizados no projeto da rede neural proposta.

2.1 Redes Neurais

De acordo com a (IBM, 2021), redes neurais são um programa ou modelo de *Machine Learning* (ML) que procura fazer decisões semelhantes ao cérebro humano.

Toda Rede neural consiste em 2 ou mais camadas de nós (neurônios artificias), sendo uma delas sempre a de saída e outra sempre a de entrada. Cada nó é conectado a outro - ou a todos em uma rede totalmente conectada -, tendo associado à cada conexão um peso. Quando o *output* (saída) de um dado neurônio exceder um limite especificado, então ele "dispara", enviando dados para a próxima camada.

[1] Localizada em: Praça Vaz Guaçu - Jardim São Paulo, São Paulo - SP, 02044-010, ou nas coordenadas: LAT. -23.4962888 LONG. -46.6200666

[2] Acessível em: https://github.com/igorzeck/MLPrecipitation

Matematicamente um nó pode ser interpretado como sendo um modelo de regressão, composto por dados de **entrada** x e **pesos** w, um ***bias*** (viés ou limite - b) e uma soma ponderada (z), tendo também uma f**unção de ativação**, ou saída ($a(z)$). Com esses parâmetros pode-se modelar o neurônio artificial mais simples possível, também chamado de *Perceptron*. Para um único neurônio, de acordo com Cristina (2021), teríamos: $z = \sum_{i=0}^{n}(w_i \cdot x_i) + b$, e para *perceptrons* teríamos uma função $a(z)$ não linear, tal que:

$$\begin{cases} a = 1, z >= 0 \\ a = 0, z < 0 \end{cases} \tag{1}$$

Vale notar que existem outras funções de ativação, dependendo do tipo de rede neural e do resultado esperado.

Como dito anteriormente, se a saída z exceder um dado limite o neurônio "dispara", enviando os dados para a próxima camada da rede, efetivamente fazendo com que a saída do nó da camada anterior se torne o *input* (entrada) da camada seguinte. O processo de passagem de dados de uma camada para a próxima define a rede como sendo do tipo *feedforward*, que será o foco deste relatório.

A medida que uma rede neural é treinada ela também tem seus pesos ajustados a fim de minimizar a chamada *loss function* ou também chamada de *cost function*. Para se minimizar essa função, temos que lidar com o chamado Problema de Minimização.

2.2 *Loss Function*

Para Zhang et al. (2023), em otimizações, a *loss function* é geralmente referida como a chamada **função objetivo**. E ainda de acordo com os autores, convencionalmente a maioria dos algoritmos estão preocupados com minimização, encontrando-se por meio de técnicas como o agoritmo do Gradiente Descendente o chamada mínimo local. Apesar de na obra citada os autores estarem descrevendo otimização sobre as lentes de algoritmos de *Deep Learning*, a definição é generalizada.

Para problemas de regressão utiliza-se a chamada *Mean Squared Error* (MSE, ou suas variantes). De acordo com Mulla (2020) A função MSE pode ser representada por:

$$MSE = \frac{1}{2} \cdot \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Sendo que, para um dado *dataset*:

- $\hat{y}$ o valor previsto por cada modelo de regressão;
- y o valor esperado (real);
- n o número de *samples* (amostras) do *dataset*;

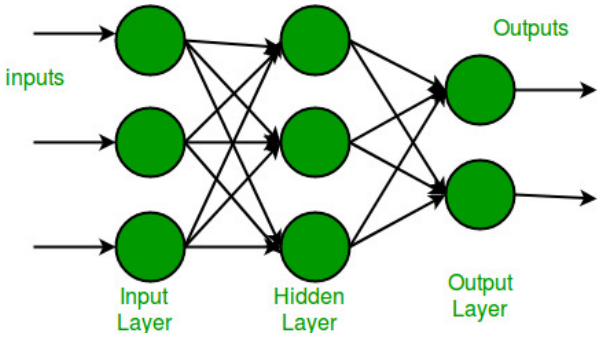
A partir da função acima pode-se utilizar técnicas de minimização de Cálculo, como por exemplo a técnica de *Gradient Descent*, base para outras técnicas de otimização, ou, como foi utilizado neste relatório, a técnica ***Adam Optmizer***.

2.3 Multi-Layer Perceptron (MLP)

Multi-Layer Percpetrons (Fgiura 3), são também chamados de *Feedforward Neural Network* - sendo este segundo o nome mais apropriado uma vez que são composto de neurônios sigmoides (e não *perceptrons*), portanto têm um *output* **entre** 0 e 1.

De acordo com GeeksforGeeks (2021) MLPs são redes com três tipos de camadas - ao contrário de modelos *perceptrons* padrão, que contém apenas dois - sendo esses tipos: (1) *Input Layers* (camada de entrada), onde cada neurônio corresponde a um *input feature*, ou seja, recebe uma variável da entrada correspondente ao *dataset* de treino; (2) *Hidden Layers* (camadas ocultas), podendo haver várias delas, sendo que cada uma contêm um número variável de neurônios, recebendo informações das camadas anteriores; (3) *Output Layers* (camada de saída), sendo aquele na qual se gera o resultado ou previsão final, contendo o número de *outputs* correspondentes à dimensão da saída. O diagrama 1 demonstra a natureza holística (totalmente conectados) das conexões entre as camadas característica de uma MLP com três camadas: 3 neurônios de *input*, 3 na camada oculta 2 no *output*.

Figura 1: **Diagrama de conexões neurais**



Fonte: GeeksForGeeks.

Ainda de acordo com os autores o MLP tem algumas características chaves, dentre elas: *Forward Propatgation*; *Loss Function*; *Backpropagation*.

2.3.1 *Forward Propagation*

É o fluxo linear e progressivo de informações de uma camada para a próxima, desde a camada de *input* até a camada de *output*. É composto pela soma ponderada $z = \sum_{i=0}^{n} w_i \cdot x_i + b$ de cada neurônio, assim como também uma função de ativação (por exemplo, a *Rectified Linear Unit* (ReLU): $a(z) = max(0, z)$).

Neste trabalho foi utilizada a **função de ativação sigmoide** (*logit*):

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

2.3.2 *Binary Classifcation Loss Function (BCE)*

Cálculo da perda feito após o *output* da rede neural. Em aprendizado supervisionado (como feito neste relatório) isso seria feito ao comparar o *output* previsto ($\hat{y}$) com o valor

verdadeiro.

Para problemas de regressão, como já mencionado, geralmente se utiliza o MSE, e neste trabalho foi utilizada uma de suas variantes, a *Binary Classifcation Loss Function* (BCE), que de acordo com o Mulla (2020) o BCE é originário da Teoria da Informação especificamente no que tange *bits* e sua perda durante o processo de transferência. Sendo assim a BCE mede o quão distante do valor verdadeiro (seja ele 0 ou 1) a predição é para cada uma das classes e então tira a média da classe toda para a obtenção da perda final. Matematicamente (GEEKSFORGEEKS, 2024):

A BCE é modelada como sendo:

$$BCE = -\frac{1}{n}\sum_{i=1}^{n}[y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)] \tag{3}$$

Com, n sendo o tamanho do *dataset* (linhas), y_i o valor real e binário (0 ou 1) e $\hat{y}_i$ sendo a probabilidade prevista. Para tal, é esperado que os dados estejam normalizados a fim de melhor representá-los como valores de probabilidade. Devido aos dados tratados pela rede neural proposta neste trabalho, utilizou-se a **BCE como *loss function***.

2.3.3 *Backpropagation*

O treino de uma MLP é feito com o objetivo minimizar a *Loss Function*, isso pode ser feito por meio da chamada *backpropagation*, sendo ela dividida em algumas etapas.

A primeira seria a de *Gradient Calculation*, que seria o cálculo do Gradiente da função de perda, ou seja:

Sendo o gradiente indicador da **direção** de maior (e também implica em menor) crescimento, para uma função qualquer $f(x, y)$ tal que $f : \mathbb{R}š : \mathbb{R}$, o gradiente é:

$$\nabla f(x, y) = (\frac{\partial f}{\partial x}(x, y), \frac{\partial f}{\partial y}(x, y)) \tag{4}$$

Ou seja, é um vetor populado pelas derivadas parciais da função $f(x, y)$ correspondentes à cada um do seus versores.

E, sendo assim, o procura-se o gradiente da *Loss Function* para poder minimizá-la, sendo que para essa função em $\mathbb{R}^{\not\vDash}$:

$$\arg\min_{(x,y)} f(x, y)$$

A segunda etapa seria a chamada *Error Propagation*, que seria a propagação do erro, camada por camada, tomando do caminho inverso (*input* $\rightarrow$ *output*).

Por último, temos o chamado *Gradiente Descent*, (Gradiente Descendente) que seria a atualização dos pesos e *biases* em si, no sentido oposto ao do gradiente, a fim de reduzir a perda L, sendo a fórmula simplificada para regressões logísticas (GEEKSFORGEEKS, 2019):

$$w = w - \lambda \cdot \frac{\partial L}{\partial w}$$

Onde, w é o peso (atual e anterior, uma vez que o algoritmo é iterativo), λ o *learning rate* - ou a porcentagem da função *Loss* subtraída do peso anterior, com o $\frac{\partial L}{\partial w}$ sendo o gradiente da função *loss* com respeito aos pesos.

A fim de otimizar iterativamente os pesos e *biases* MLPs utilizam métodos de otimização, como o chamado *Adam Optmizer* - utilizado nesse artigo. O *Adam Optmizer* é uma extensão do *Root Mean Square Propagation* (RMSprop) incorporando um *momentum* ao *learning rate*, tornando-o adaptável e mais eficiente após cada iteração do treino.

O *momentum* é, no contexto do algoritmo de gradiente, utilizado "[...] para acelerar o processo de Gradiente Descendente ao incorporar uma média móvel exponencial de gradientes anteriores"(tradução nossa) (GEEKSFORGEEKS, 2020). No seu modo mais simplificado: $w_{t+1} = w_t - \lambda m_t$, onde o m_t é a média móvel dos gradientes para o tempo t e, o w_t w_{t+1} seria o peso para o tempo correspondente. Por sua vez, podemos expandir o termo m_t:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\frac{\partial L}{\partial w_t} \tag{5}$$

com o termo β sendo o parâmetro do *momemtum* (geralmente 0.9) e o gradiente sendo para a função *Loss* com respeito ao peso para o tempo t.

A partir do *momentum* podemos calcular RMSprop - dos quais o cálculo foge do escopo desse trabalho, uma vez que são integrados ao cálculo do *Adam Optimizer*.

No algoritmo sintetizado neste trabalho foi utilizado no parâmetro *optimizer* uma instância do **otimizador Adam** na função "compile"do modelo. Nesse caso, por se tratar de uma instância padrão, são utilizados os **valores padrões** da classe Adam[3]: os valores de β_1 e β_2 foram os padrões que são 0.9 e 0.999 respectivamente, e o ϵ - utilizado para evitar divisões por zero - teve o valor de 10^{-7}.

Para o otimizador Adam se utilizam as estimativas de primeiro (m_t média) e segundo (v_t variância) momento, além do passo de correção de viés, uma vez que ambas estimativas tendem a ser enviesadas para zero. Segue abaixo os cálculos para ambas as estimativas:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\frac{\partial L}{\partial w_t}$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\left(\frac{\partial L}{\partial w_t}\right)^2$$

E para a correção de viés:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Dessa forma, o atualização final dos pesos é:

$$w_{t+1} = w_t - \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Os cálculos de retro-propagação são feitos x vezes, onde x é o número de épocas (*epochs*), com cuidado com para evitar hiper-ajuste (ocorre quando dados de treino muito homogêneos) ou sob-ajuste (ocorre quando os dados de treinos não são suficientes para a previsão).

[3]Valores padrões podem ser conferidos em: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Ada

2.4 Dataset do INMET

Os dados utilizados neste relatorio foram extraídos em formato *Comma Separeted Values* (CSV), com os valores separados por ";"e as primeiras 10 colunas contendo informações operacionais do *dataset*. Manuseou-se os dados através da biblioteca Pandas da linguagem Python, e a limpeza do *dataset* foi feita da seguinte forma:

1. A coluna "Data"foi convertida para o formato *datetime*, e as colunas "Data Medicao"e "Hora Medicao"foram excluídas. Passou-se a coluna "Data"como índice do *Dataframe df* representante do *dataset* - tudo isso para a criação de uma sequência temporal de fácil utilização e plotagem pelo Pandas.

2. Obteve-se a matriz de correlação (de Pearson, que é o padrão da função), sendo esta plotada como um Mapa de Calor utilizando-se a biblioteca Seaborn e Matplotlib.

3. Procurou-se então os dados de mais correlação para com a variável alvo (*target*) "PRECIPITACAO TOTAL, HORARIO(mm)": Excluiu-se todos os dados que tem menos de 10% de correlação com a variável alvo.

4. Salvou-se os resultados em um arquivo CSV para o treino do modelo.

As variáveis selecionadas com os valores de correlação acima do mínimo definido foram[4]:

(A) "PRECIPITACAO TOTAL, HORARIO(mm)"(**Variável target**);

(B) "TEMPERATURA DO PONTO DE ORVALHO(°C)";

(C) "TEMPERATURA ORVALHO MAX. NA HORA ANT. (AUT)(°C)";

(D) "TEMPERATURA ORVALHO MIN. NA HORA ANT. (AUT)(°C)";

(E) "UMIDADE REL. MAX. NA HORA ANT. (AUT)(%)";

(F) "UMIDADE RELATIVA DO AR, HORARIA(%)";

(G) "VENTO, RAJADA MAXIMA(m/s)";

Ao longo deste trabalho, as variáveis listas serão referenciadas pelos suas letras (A-G) na lista acima, incluindo em gráficos.

3 METODOLOGIA

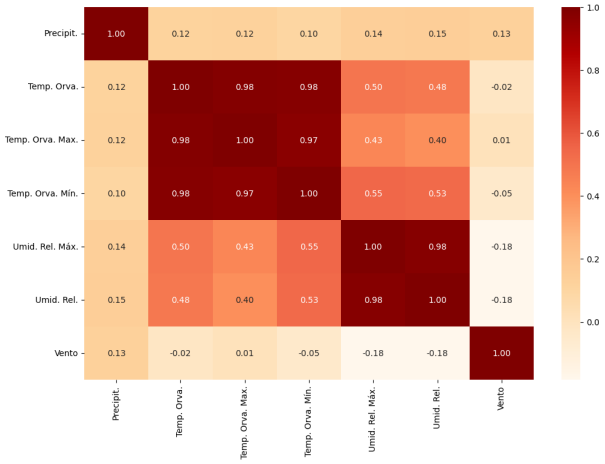
3.1 *Exploratory Data Analysis*

Os dados foram solicitados por meio do Banco de Dados Meteorológicos (BDMEP) do INMET[5] e solicitados com os seguintes parâmetros:

- Separados por ponto e vírgula (;);
- Frequência horária;

[4]Dicionário acessível em: https://portal.inmet.gov.br/glossario/glossario
[5](acessível em https://bdmep.inmet.gov.br/)

Figura 2: **Mapa de Calor das variáveis relevantes**



Fonte: elaborado pelo autor com base nos dados do INMET.

- De estações automáticas;
- De Abrangência regional;
- No período compreendido por 10/10/2010 a 25/05/2025;
- Região sudeste;
- Todas as variáveis disponíveis;
- Selecionada a opção ”SAO PAULO - MIRANTE (1) - [SP]”para os dados da estação de escolha;

Foi realizado um processo de *Extract Transform Load* (ETL) dos dados, assim como também um rápida *Exploratory Data Analysis* (EDA) para se familiarizar com o *dataset*. Por meio desse EDA foi possível aferir algumas caratcterísticas gerais do *dataset*, ele contém 128.208 entradas horárias, sendo que destas 2.616 contém algum valor ausente, além disso o *dataset* contém 18 colunas (contando com o *target*).

Foram então retirados os valores que tinham uma correlação absoluta com a variável *target* menor do que 0,1, sobrando apenas 7 variáveis relevantes 2.

Após uma exploração simples do *dataset* esse foi exportado contendo apenas as colunas relevantes para um arquivo CSV.

3.2 *Elaboração e treino da rede neural*

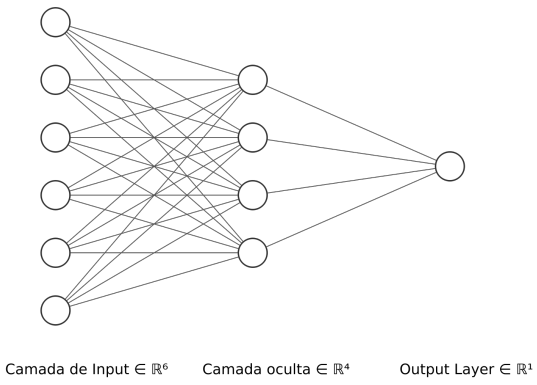
Os dados extraídos do *dataset* gerado durante o processo de EDA foram normalizados, tal que $\tilde{x_i} = \frac{x_i - \overline{x}}{\sigma}$, com $\tilde{x_i}$ sendo os valores normalizados, x_i os valores originais, $\overline{x}$ a média da variável x, e σ sendo o desvio padrão. A etapa de normalização foi feita para todas as colunas, **exceto para a coluna de precipitação**. Isso se deve ao fato de assim facilitar a conversão dessa coluna para uma coluna de valores dicotômica, com dois possíveis valores 0 - para quando não há registro de precipitação - e 1 - para quando há registro de precipitação.

Após isso é feita a separação e filtragem dos dados para treino e o teste. Os dados são separados em dois conjuntos, o primeiro, X, contendo os *features* que serão utilizados de

input tanto para o treino quanto para o teste, o segundo é o Y, que contém os dados *target* de precipitção, utilizados no *output*. É então feita um partição dos dados de treino e teste baseado nesses dois conjuntos por meio da função "train_test_split"da biblioteca Scikit-learn. São utiliazdos 20% dos dados para teste e os restantes 80% para treino. Para fins de teste reprodutibilidade foi utilizada uma *seed* de 42 para o parâmetro "random_state".

Após a separação dos dados foi então criado o modelo. Ele contém 3 camadas de neurônios: (1) Camada de Input (com 6 entradas); (2) Camada oculta, com 4 neurônios e método de ativação sigmoide; (3) 1 neurônio de saída no *output* com ativação também sigmoide 3.

Figura 3: **Diagrama da rede neural**



Fonte: diagrama elaborado pelo autor através do site: https://alexlenail.me/NN-SVG/index.html (acesso em: 08/06/25).

Após isso, foi foi criado um objeto de "early_stopping"para terminar o treino caso em 5 iterações consecutivas haja piora no parâmetro *Loss*. O modelo é então compilado utilizando a função Loss *Binary Cross-Entropy* o Otimizador Adam.

O treino em si só é feito caso não haja um modelo já salvo no diretório raiz. Se esse for o caso os parâmetros do treino são[6]: "*epochs*"como 100, ou seja, são 100 grandes iterações para todo o conjunto X e Y fornecidos; "batch_size"sendo o tamanho da amostra por atualização da função gradiente. Em geral, números menores geram resultados mais acurados; "validation_data"para os valores de treino anteriormente seccionados; *"callbacks"* recebendo o objeto "early_stopping"supracitado, e com o parâmetro "restore_best_weights"como *True*;

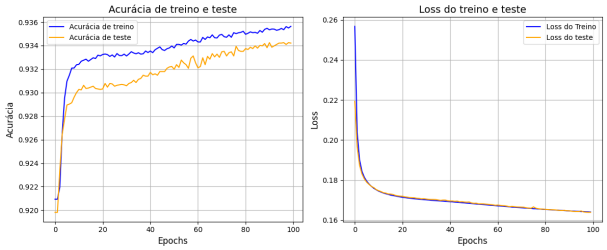
4 ANÁLISE

O modelo treinado resultante obteve acurácia geral de aproximadamente 93% e *Loss* mínimo de aproximadamente 0.16. Pode-se observar a tendência histórica da acurácia e do *Loss* historicamente (Figura4):

Os pesos também podem ser extraídos da rede neural, e a partir deles podemos criar uma matriz para o cálculo dos valores previstos:

[6]Significado dos parâmetros encontrado aqui (acesso em: 08/06/2025)

Figura 4: **Histórico do Treino e Teste**



Fonte Elaborado ao longo deste relatório.

$$W_1 = \begin{bmatrix} 0.97634286 & -1.9522462 & 0.34698686 & -2.7568676 & -0.6865847 & -0.16645359 \\ -1.433105 & -2.2095802 & 4.080188 & -0.6881494 & -1.051062 & -0.31405678 \\ -1.908303 & -2.7503285 & 4.137449 & 1.6544156 & -0.8605119 & -0.34463218 \\ -1.1312401 & 1.5943722 & -0.7186267 & -0.0605193 & 2.9796646 & 0.070917726 \end{bmatrix}$$

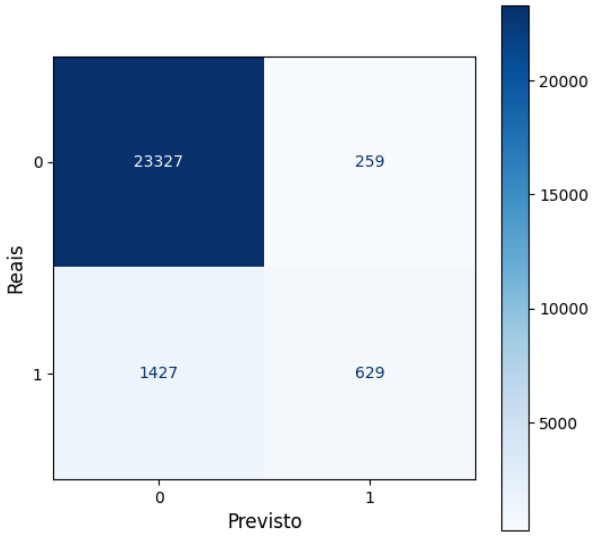
$$B_1 = \begin{bmatrix} 1.531187 \\ 0.24852464 \\ -0.39425907 \\ -2.6061049 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} -3.4306805 \\ -3.2021632 \\ -4.5189514 \\ 3.771724 \end{bmatrix}$$

$$B_2 = \begin{bmatrix} 0.87639076 \end{bmatrix}$$

Onde $Y_1 = W_1 \cdot X + B_1$, e $Y_2 = O(X) = W_2 \cdot Y_1 + B_2$, gerando assim uma previsão final. Os valores de previsão final podem ser verificados na figura abaixo (Figura 5):

Figura 5: **Matriz de confusão**



Fonte: Elaborado pelo autor.

Pode-se verificar na matriz de confusão que tanto para valores 0 - sem precipitação - quanto para valores 1 - com precipitação - o modelo teve alta acurácia em suas previsões, no entanto, houve uma quantidade considerável de erros falso negativos.

5 CONSIDERAÇÕES FINAIS

Conclui-se que a rede neural MLP treinada apesar de simples em sua configuração foi efetiva em sua previsão de valores de precipitação - dados todas as demais variáveis conhecidas - atingindo uma acurácia muito boa (93% de acurácia) para a previsão de valores de precipitação dados outras 6 variáveis altamente correlacionados. Há no entanto, limitações. No que diz respeito ao modelo, ele não foi necessariamente feito para a previsão de valores de sequência longa, e portanto a previsão de mais uma hora simultaneamente para um valor de precipitação ou até mesmo a previsão de todas as variáveis para um ou mais etapas a diante se demonstraria problemático para o modelo, uma vez que o ele não foi otimizado para lidar com dados sequenciais tal como, por exemplo, uma *Recurrent Neural Network* (RNN) poderia.

Referências

CRISTINA, Stefania. **Calculus in Action: Neural Networks**. [S.l.: s.n.], ago. 2021. Disponível em: <https://machinelearningmastery.com/calculus-in-action-neural-networks/>.

GEEKSFORGEEKS. **Binary Cross EntropyLog Loss for Binary Classification**. [S.l.]: GeeksforGeeks, mai. 2024. Disponível em: <https://www.geeksforgeeks.org/binary-cross-entropy-log-loss-for-binary-classification/>. Acesso em: 8 jun. 2025.

_____. **Gradient Descent algorithm and its variants**. [S.l.: s.n.], fev. 2019. Disponível em: <https://www.geeksforgeeks.org/gradient-descent-algorithm-and-its-variants/>. Acesso em: 8 jun. 2025.

_____. **Multi-Layer Perceptron Learning in Tensorflow**. [S.l.: s.n.], nov. 2021. Disponível em: <https://www.geeksforgeeks.org/multi-layer-perceptron-learning-in-tensorflow/>. Acesso em: 8 jun. 2025.

_____. **What is Adam Optimizer?** [S.l.: s.n.], out. 2020. Disponível em: <https://www.geeksforgeeks.org/adam-optimizer/>. Acesso em: 8 jun. 2025.

IBM. **What is a Neural Network?** [S.l.: s.n.], out. 2021. Disponível em: <https://www.ibm.com/think/topics/neural-networks>. Acesso em: 25 mai. 2025.

MULLA, Mohammed Zeeshan. **Cost, Activation, Loss Function|| Neural Network|| Deep Learning. What are these?** [S.l.: s.n.], mai. 2020. Disponível em: <https://medium.com/@zeeshanmulla/cost-activation-loss-function-neural-network-deep-learning-what-are-these-91167825a4de>.

ZHANG, Aston et al. **Dive into deep learning**. [S.l.]: Cambridge University Press, 2023.